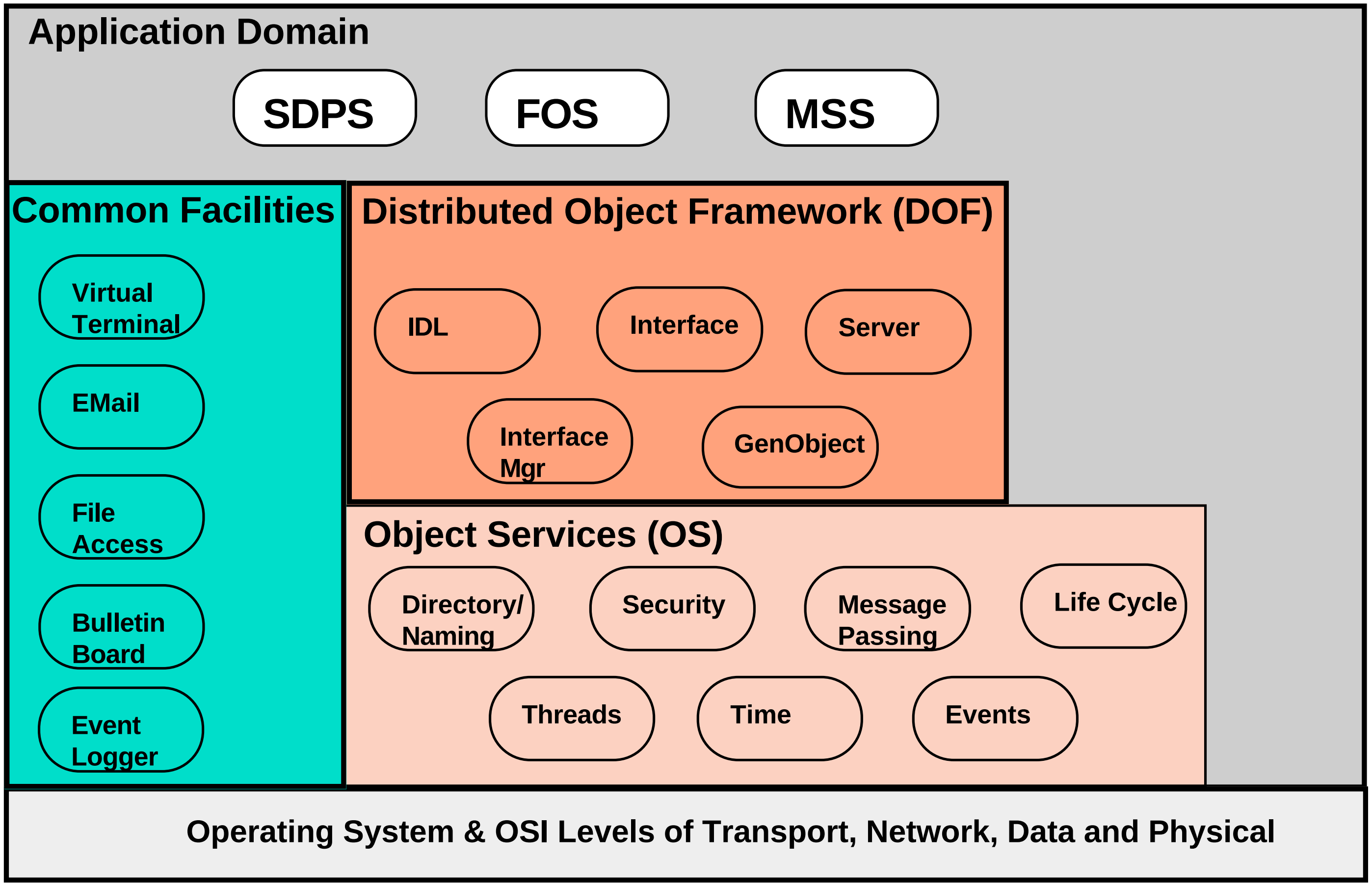
CSS Services

Naveen Hota

18 January 1995

CSS Services



Roadmap

- Common Facilities
- Object Services (OS)
- Distributed Object Framework (DOF)
  - Client Server Concepts
  - Interface
  - Client/Server Application Development
- Communication Mechanisms
  - Message Passing
  - Deferred Synchronous Message Passing Scenario
  - Communication Mechanisms Summary

Common Facilities

Common facilities are high level services with uniform semantics that are shared across applications.

- File Access - provides file transfer and remote file access capabilities across hosts in a network environment. This service includes kerberized ftp.
- E-Mail - provides the functionality to manage electronic mail messages for M&O staff and applications.
- Bulletin Board - provides a forum for sharing ECS related information to the users.
- Virtual Terminal - provides users the capability to remotely log into designated ECS hosts. This service includes kerberized telnet and X.
- Event Logger - enables applications to record information to designated log files.

Object Services

- Low level building blocks
- ECS applications interact with them transparently through DOF
- Object Services provided in ECS are
  - Directory/Naming
  - Security
  - Threads
  - Time
  - Event
  - Lifecycle
  - Message Passing

Object Services (cont.)

- Directory/Naming
  - Host lookup and binding information.
  - Location transparency.
  - Replication, distribution and caching.
  - BIND (DNS), GDS (X.500), and CDS (OSF) namespaces.
  - Extend the namespace with additional application specific information.
  - Examples:

    Locate ECS services/resources.

    FOS Planning and Scheduling processes registering interest in receiving updated schedules from the Resource model.
- Time
  - Provides mechanisms to keep host clocks in network environments approximately in sync.
  - Example: Timestamps in History Logs.

Object Services (cont.)

- Security
  - Protects distributed resources through

    Authentication

    Authorization

    Data integrity

    Data privacy
  - Provides customized authentication and authorization class libraries for ACL management.
  - Examples: ECS Resource Access, User Login.
- Threads
  - Provides an efficient and portable way for asynchronous and concurrent processing (Posix 1003.4a).
  - Example: Multiple server threads to process client calls.

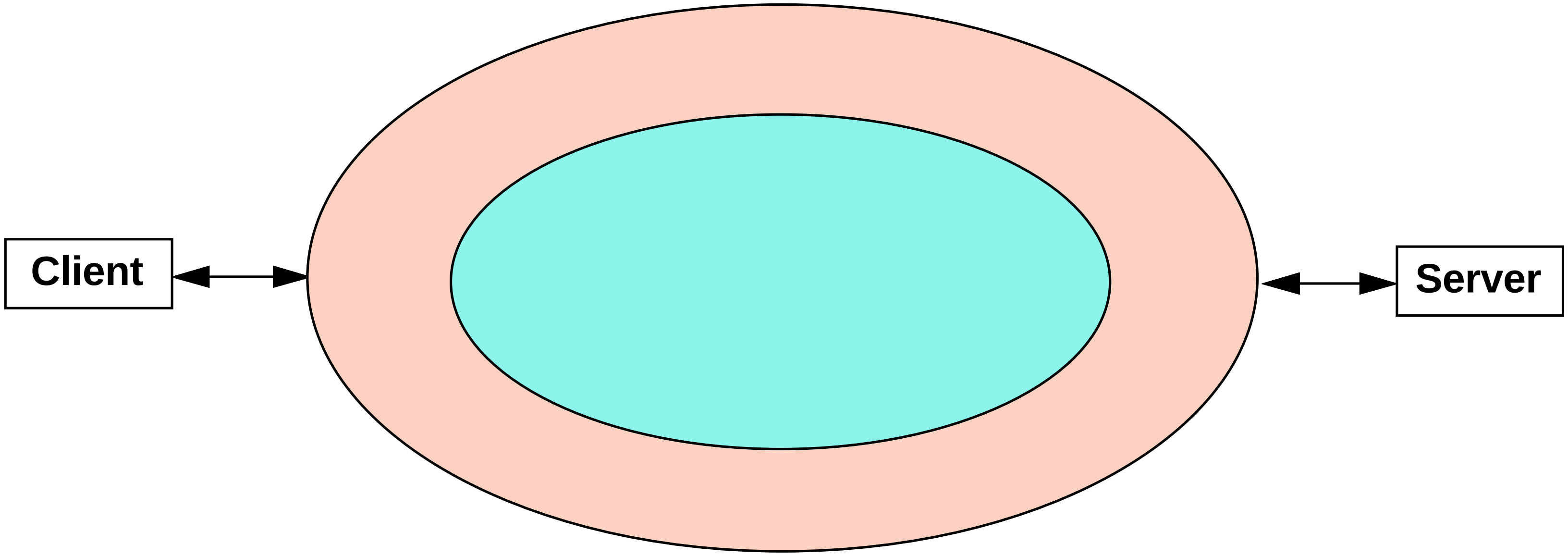
Object Services (cont.)

- Event
  - Provides asynchronous communication between objects with clear de-coupling.
  - Examples: Short broadcasts of system messages.
- Lifecycle
  - Provides client functionality to transparently access inactive services.
  - Examples: Startup of application servers.
- Message Passing
  - Provides asynchronous and deferred synchronous message passing between processes.
  - Examples: Real time telemetry data between the FOS DECOM and the User Interface.

Roadmap

- Common Facilities
- Object Services (OS)
- Distributed Object Framework (DOF)
  - Client Server Concepts
  - Interface
  - Client/Server Application Development
- Communication Mechanisms
  - Message Passing
  - Deferred Synchronous Message Passing Scenario
  - Communication Mechanisms Summary

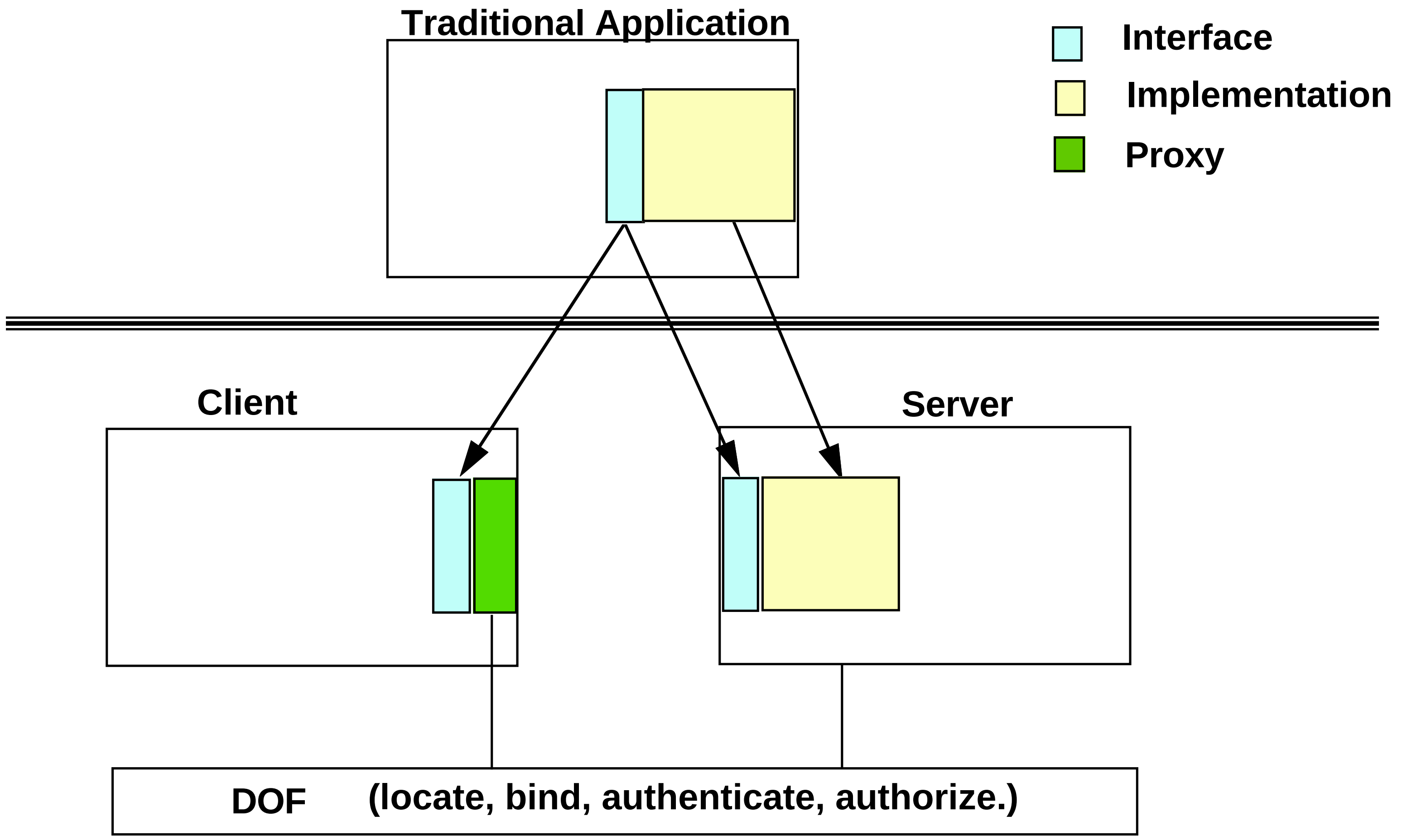
Distributed Object Framework



Distributed Object Framework (cont.)

- **Object Oriented (OO) layer on top of the Object Services for developing the ECS client/server applications.**
- **Implementation via RPCs.**
- **Clear separation between interface and implementation.**
- **Isolates the application developers from low level communication programming.**
- **Provides OO libraries from which the applications inherit functionality.**

Migration to Client/Server



Traditional Application

🟦 Interface

🟨 Implementation

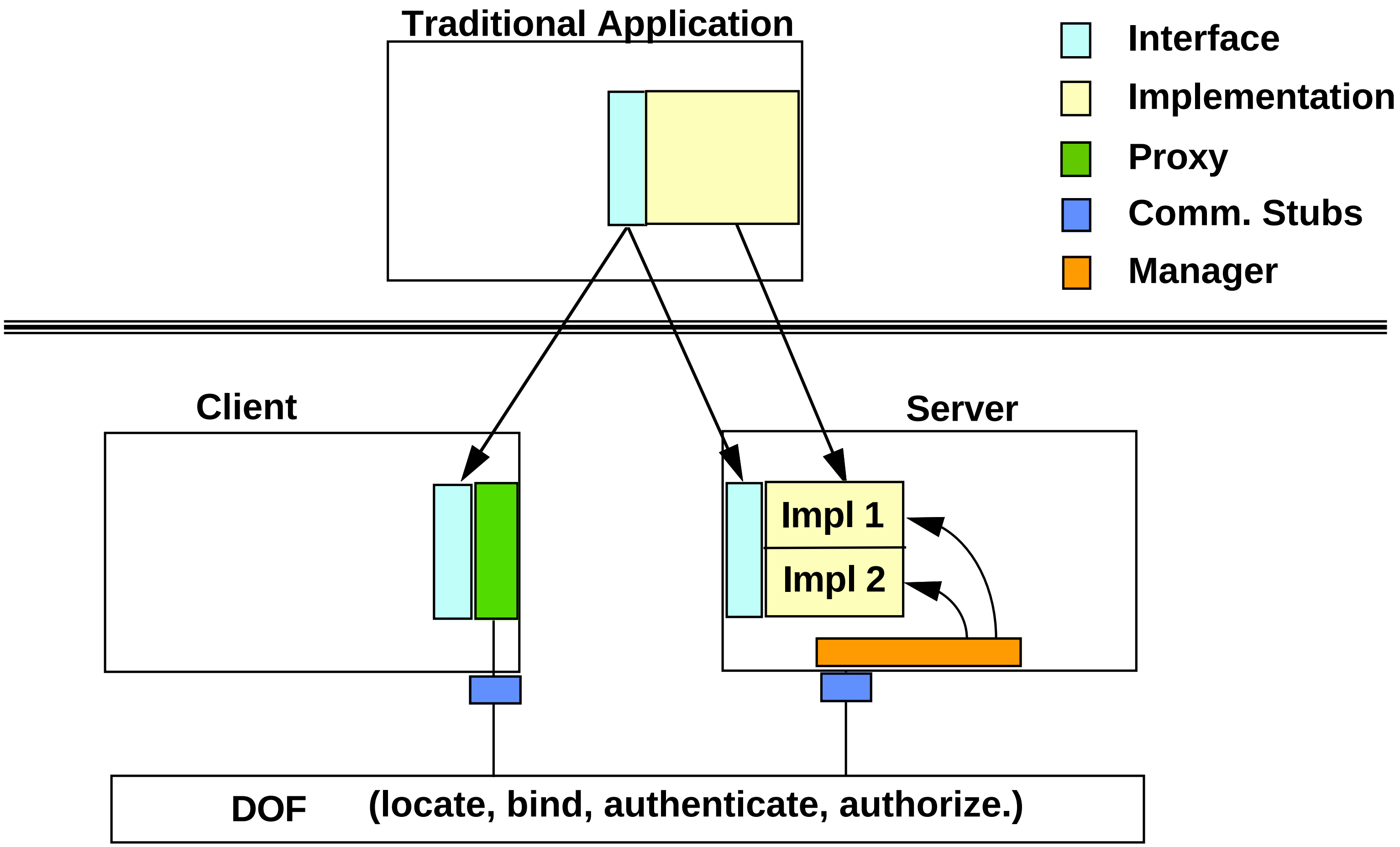
🟩 Proxy

Client

Server

DOF (locate, bind, authenticate, authorize.)

Migration to Client/Server (cont.)



Traditional Application

🟦 Interface

🟨 Implementation

🟩 Proxy

🟦 Comm. Stubs

🟧 Manager

Client

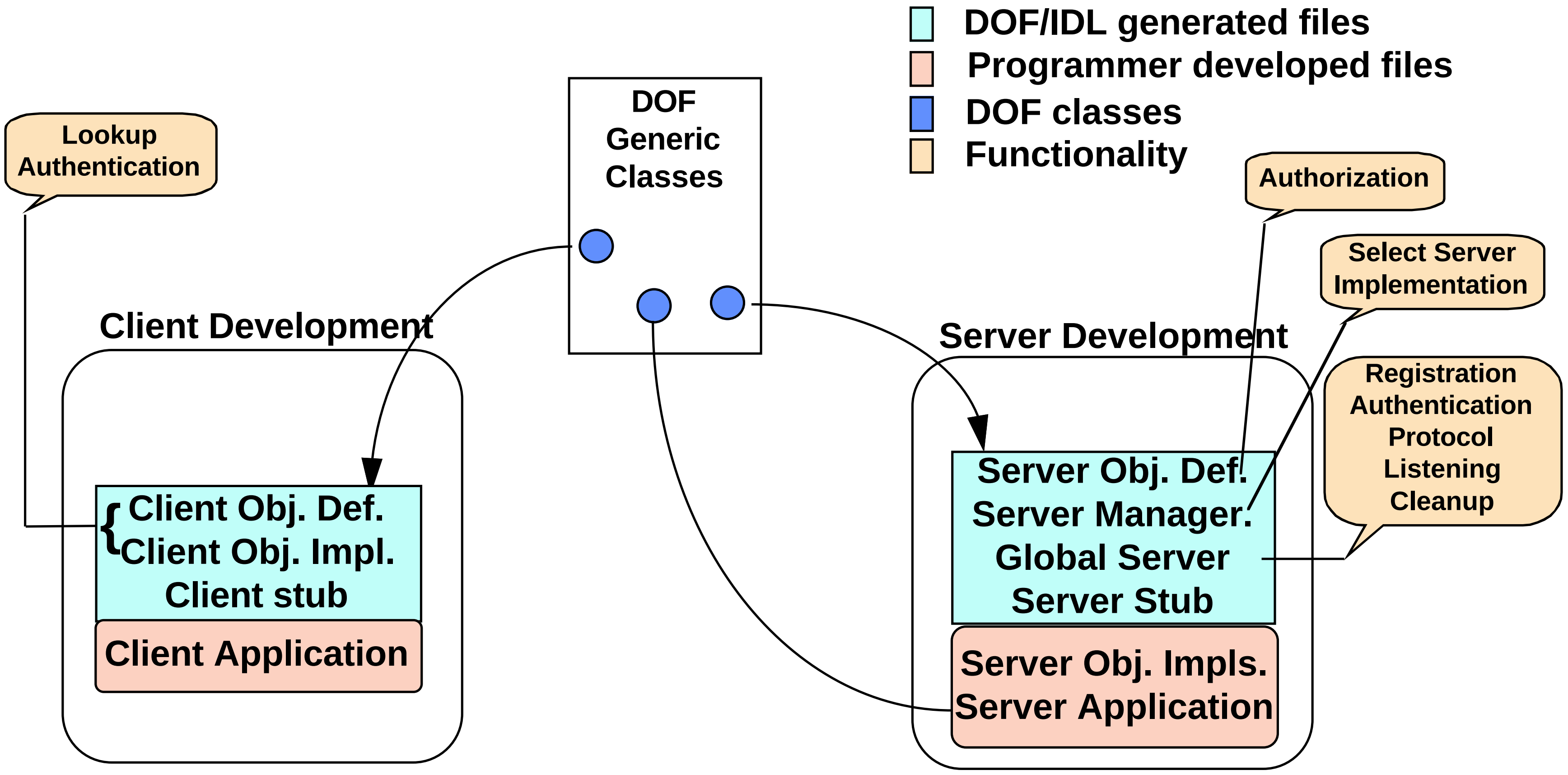
Server

Impl 1

Impl 2

DOF (locate, bind, authenticate, authorize.)

Client/Server Application Components



🟦 DOF/IDL generated files

🟥 Programmer developed files

🟦 DOF classes

🟧 Functionality

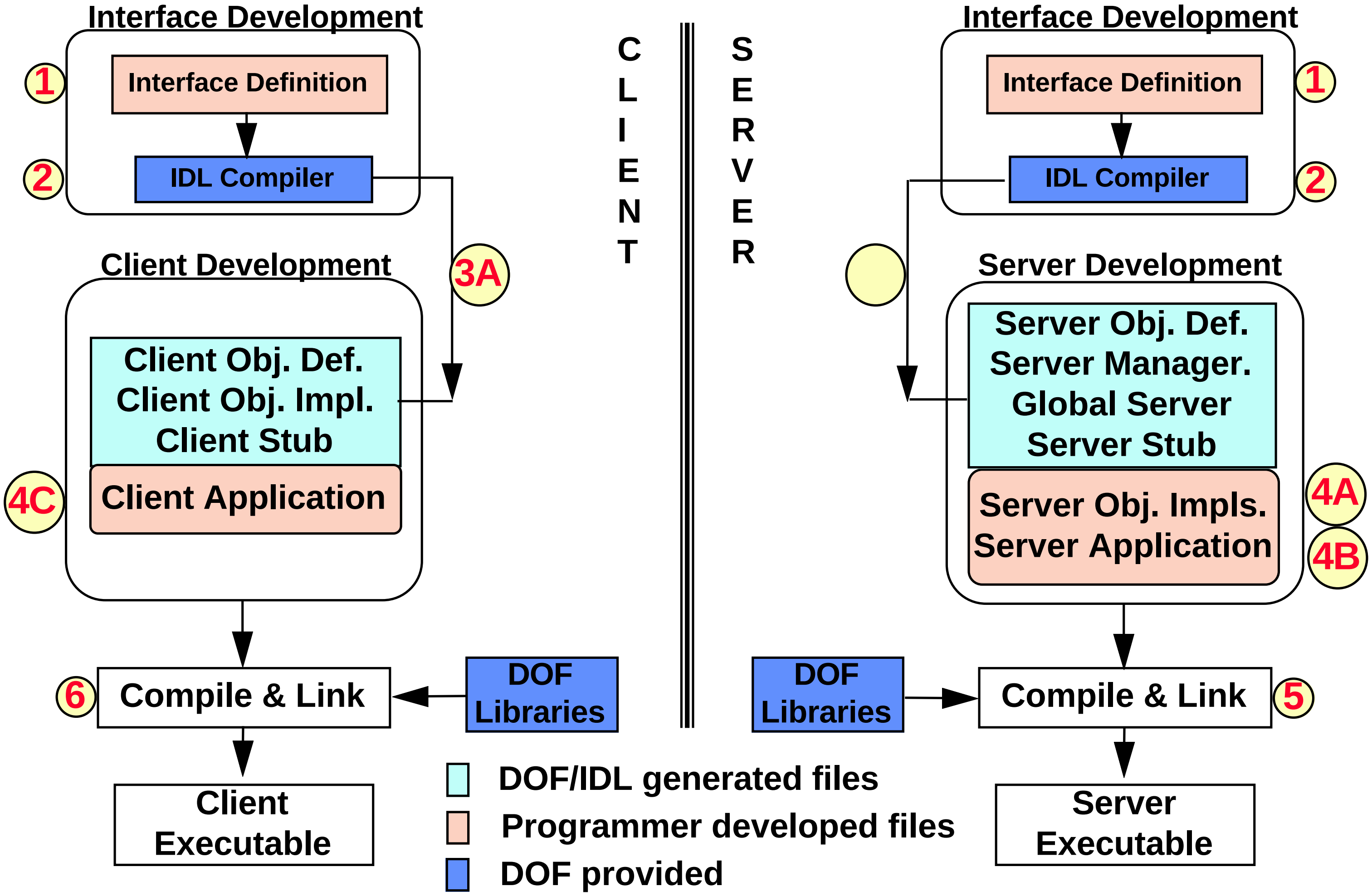
DOF Interface

- Provides generic classes for distributed client/server applications.
- Uses Object Services: Naming, Security, Threads and Time.
- Client class inherits from “Interface” class.
    - Provides location and security preferences.
- Server Implementations inherit from “InterfaceMgr” class.
    - Provides Authorization.
- Server application uses a global instance of the Server object.
    - This object transparently interacts with the object services and provides functionality for object registration, protocol selection, naming and security preferences, cleaning up and listening for client requests.
    - It also uses a Server Manager to select a server implementation when multiple implementations of the server object exists.

DOF Benefits

- Location transparency
- Invocation independence
- Network based security
- Heterogeneous & Interoperable
- Supports OO paradigm
- Generic class libraries with default behavior
- Customizable by application developer for specialized behavior
- Transparent interaction with the underlying Object Services

Writing Client/Server Applications



C L I E N T

Interface Development

1 Interface Definition

2 IDL Compiler

3A

Client Development

Client Obj. Def.
Client Obj. Impl.
Client Stub

4C Client Application

6 Compile & Link

DOF Libraries

Client Executable

S E R V E R

Interface Development

Interface Definition 1

IDL Compiler 2

Server Development

Server Obj. Def.
Server Manager.
Global Server
Server Stub

Server Obj. Impls.
Server Application 4A 4B

DOF Libraries

Compile & Link 5

Server Executable

▢ DOF/IDL generated files

🟧 Programmer developed files

🟦 DOF provided

Writing Client/Server Applications (cont.)

- Define the client/server interface in Interface Definition Language (IDL) to specify remote procedures.
- Compile the IDL file to generate the following stub files containing
  - Client object definition
  - Client object implementation
  - Client stub for communications
  - Server object definition
  - Server Manager
  - A Global Server object
  - Server stub for communications

Writing Client/Server Applications (cont.)

- Develop the server implementation
  - Identify all the different implementations the server is going to support.
  - Implement member functions for each implementation.
  - Customize authorization [optional].
  - Create and populate Access Control Lists (ACL) with entries containing principal/group and associated permissions. This can be done programmatically or can be read from an ACL Database file [optional].

Writing Client/Server Applications (cont.)

- Develop the server application
  - Construct an instance of the server object.
  - Register the server object with the Global Server object.
  - Identify the protocols (tcp/udp) to be used in servicing the requests [optional].
  - Specify the maximum number of threads the service can run to execute the user specified services concurrently [optional].
  - Register the authentication information [optional].
  - Establish separate server identity [optional].
  - Register the binding information for each combination of the interface name , protocol, and implementation in the local (CDS) namespace [optional].

Writing Client/Server Applications (cont.)

   - Register the binding information in foreign namespaces [optional].
   - Start a separate thread and go into a listen loop and wait for incoming requests.
   - Wait for the thread to finish or wait for a shutdown message or user interrupt (kill signal).
   - Remove the binding information from namespace(s) and exit gracefully.
- Compile the server main and the server implementation.
- Link them with the stub files and DOF libraries to generate the executable.
- Run the executable or alternatively, let the Lifecycle Service run it on demand.

Writing Client/Server Applications (cont.)

- Develop the client
  - Client object provides several constructors to identify servers.

    Interface

    CDS name

    Host address and protocol

    Object reference

    Binding handle
  - Instantiate local client object to locate and access server object through one of the above constructors.
  - Set client security preferences [optional].
  - Invoke methods in the local client object.
- Compile and link client source with IDL generated client class, runtime stubs, and DOF class libraries.

Writing Client/Server Applications (Summary)

- Define the interface between the client and the server.
- Develop the server implementation(s).
- Develop the server application that initializes the server object(s).
- Develop the client application that invokes methods in the local client object (proxy).
- Compile and Link the object files with DOF libraries to produce server and client executable.
- Run the server executable which listens continuously.
- Run the client application to create client object and make calls to it.

Roadmap

- Common Facilities
- Object Services (OS)
- Distributed Object Framework (DOF)
  - Client Server Concepts
  - Interface
  - Client/Server Application Development
- Communication Mechanisms
  - Message Passing
  - Deferred Synchronous Message Passing Scenario
  - Communication Mechanisms Summary

Communication Mechanisms

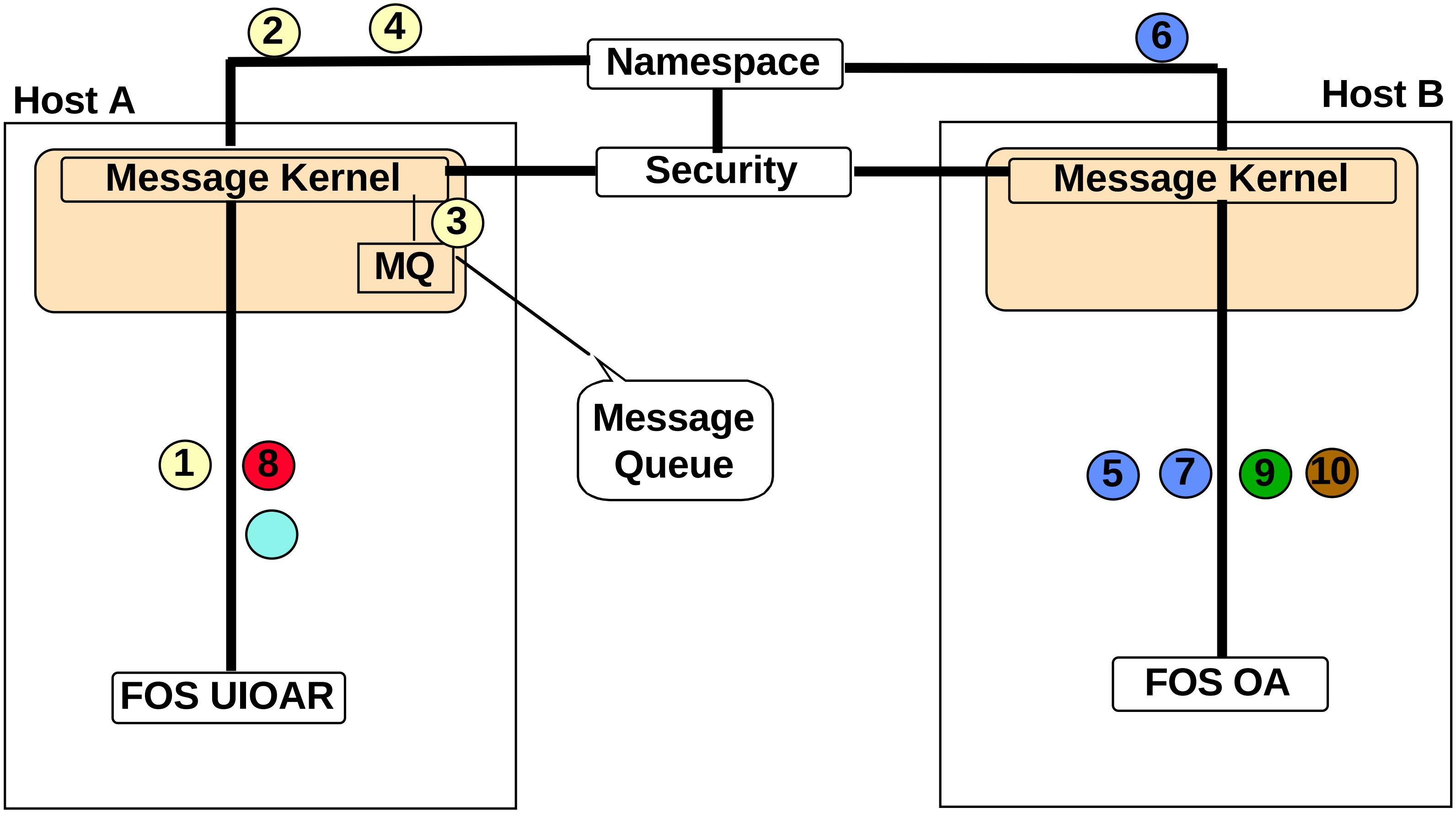
Three ways of transferring data from “Sender” to “Receiver”

- Synchronous
  - Sender waits for Receiver to complete processing and return the results (blocking).
  - Provided by DOF.
  - Used for normal remote method invocation.
- Asynchronous
  - Sender makes a call to send data to the Receiver and continues processing (non blocking). No results are returned.
  - Provided by Message Passing Service.
  - Used for transfer of large data.
- Deferred synchronous
  - Same as asynchronous, except data is processed at the remote end and the sender can receive the result at a later time (non blocking).
  - Provided by Message Passing Service.
  - Used for process intensive remote applications.

Message Passing

- Need determined from extensive discussions with FOS.
- Channels data across processes in a heterogeneous environment through intermediary message queues.
- Supports Asynchronous and Deferred Synchronous mechanisms.
- Coexists with DOF and makes use of Naming and Security services.
- Sender and Receivers are coupled as opposed to Event service.
- Guaranteed delivery of data.
- Supports priorities & persistence.
- Implementation using COTS/custom.

Deferred Sync. Message Passing Scenario



Deferred Sync. Message Passing Scenario (cont.)

- FOS User Interface Off-line Analysis Request (UIOAR) process requests the local message kernel to create a message queue with a unique name “uioar”.
- The message kernel consults with Namespace to check if “uioar” message queue already exists.
- The message kernel creates a new message queue (“uioar” doesn’t exist yet).
- The message kernel registers the newly created “uioar” message queue in the Namespace. Returns a handle of “uioar” to the UIOAR process.
- FOS Off-line Analysis (OA) process requests its local message kernel to get the address of the “uioar” message queue.
- The local message kernel consults with the Namespace and returns a handle of the existing “uioar” message queue to the OA process.

Deferred Sync. Message Passing Scenario (cont.)

● **OA process registers interest with the “uioar” message queue in receiving any messages that are address to it.**

● **The UIOAR process sends a Deferred Synchronous message addressed to the OA process and gets a unique ticket from the message kernel to redeem results at a later time. The UIOAR continues processing.**

9 **The message kernel sends the data to the OA process along with the same unique ticket it sent to the UIOAR (“uioar”).**

● **The OA process analyses the data at a later time and returns the result and the unique ticket back to the message kernel.**

11 **The UIOAR process periodically polls the message kernel to check if the results are available and obtains them by presenting the unique ticket.**

Communication Mechanisms Summary

	Synchronous (DOF)	Asynchronous (Msg Passing)	Deferred Synchronous (Msg Passing)	Events (Object Services)
Blocking	yes	no	no	no
Return results	yes	no	yes	no
Designated receivers	yes	yes	yes	no
Multiple receivers	no	yes	no	yes
Guaranteed	yes	yes	yes	n/a
Acknowledgment	yes	yes	yes	no
Argument types	supported types	byte stream	byte stream	byte stream
Callbacks	n/a	maybe	maybe	no
Priorities	n/a	yes	yes	no
Store/forward	n/a	yes	yes	yes
Large data	yes	yes	yes	no
Receiver listens	yes	yes	yes	yes
Receiver monitors	no	yes	yes	no
Process intensive	no	n/a	yes	n/a